Exploiting Redundancy for Timeliness in TCP Boston

Azer Bestavros
best@cs.bu.edu

Gitae Kim
kgtjan@cs.bu.edu

Computer Science Department
Boston University
Boston, MA 02215

Tel: (617) 353-9726
Fax: (617) 353-6457

Abstract

While ATM bandwidth-reservation techniques are able to offer the guarantees necessary for the delivery of real-time streams in many applications (e.g. live audio and video), they suffer from many disadvantages that make them inattractive (or impractical) for many others. These limitations coupled with the flexibility and popularity of TCP/IP as a best-effort transport protocol have prompted the network research community to propose and implement a number of techniques that adapt TCP/IP to the Available Bit Rate (ABR) and Unspecified Bit Rate (UBR) services in ATM network environments. This allows these environments to smoothly integrate (and make use of) currently available TCP-based applications and services without much (if any) modifications. However, recent studies have shown that TCP/IP, when implemented over ATM networks, is susceptible to serious performance limitations. In a recently completed study, we have unveiled a new transport protocol, TCP Boston, that turns ATM's 53-byte cell-oriented switching architecture into an advantage for TCP/IP. In this paper, we demonstrate the real-time features of TCP Boston that allow communication bandwidth to be traded off for timeliness. We start with an overview of the protocol. Next, we analytically characterize the dynamic redundancy control features of TCP Boston. Next, We present detailed simulation results that show the superiority of our protocol when compared to other adaptations of TCP/IP over ATMs. In particular, we show that TCP Boston improves TCP/IP's performance over ATMs for both network-centric metrics (*e.g.*, effective throughput and percent of missed deadlines) and real-time application-centric metrics (e.g., response time and jitter).

Keywords: ATM networks; TCP/IP; Adaptive Information Dispersal Algorithm; real-time communication; congestion control; performance evaluation.

1 Introduction

The increasing reliance on the Transmission Control Protocol (TCP) [28]—a reliable transport protocol that uses a window-based flow and error control algorithm on top of the Internet Protocol (IP) layer—as the transport protocol of choice for many Internet/Intranet applications that require real-time (or near real-time) performance is evident. One reason for this is the mushrooming use of the TCP/IP-based World Wide Web as an infrastructure for large-scale distributed information systems, the access of which may be essential for applications that are subject to soft or firm timing constraints. The proliferation of TCP/IP is clearly manifest in the vast array of services and applications that rely on TCP's robust functionality and its hiding of the underlying details of networks of various scales and technologies, from Local Area Networks (LANs) to Wide Area Networks (WANs), and from Ethernets to Satellite networks.

Recently, the premise of high-speed communication using Asynchronous Transfer Mode (ATM) technology has raised the hopes for a better support of real-time (or near real-time) applications. However, the inability of this technology to effectively support TCP/IP has dampened such hopes. The ATM technology is a connection oriented, 53-byte cell-based transport technology, which offers high-speed switching for both LANs and WANs. ATM is designed to support a variety of applications with diverse requirements using both *bandwidth-reservation* and *best-effort* techniques.

Motivation: While bandwidth-reservation techniques are able to offer the guarantees necessary for the delivery of real-time streams in many applications (*e.g.* live audio and video), they suffer from many disadvantages that make them inattractive (or impractical) for many others.

First, the self-similarity (large and unpredictable variability) of real-time traffic (*e.g.* MPEG video streams [22, 21]) makes bandwidth reservation techniques inherently inefficient in their use of bandwidth, especially in applications that require the concurrent transmission of real-time data (*i.e.* subject to absolute and relative temporal constraints [30]) from/to a large number (potentially thousands) of sources/destinations. Setting up virtual circuits with guaranteed bandwidth for such connections is simply impractical, particularly given that the nature of data transmission (*e.g.* rates and sources) may vary wildly. Example applications include real-time network monitoring and visualization, large-scale group simulations, and *etc.*

Second, the setting up and tearing down of virtual circuits with guaranteed-bandwidth are expensive operations that may not be justified for short-lived connections. Such connections, involving short file transfers (*e.g.* using HTTP or FTP) are likely to constitute the majority of inter/intranet traffic in the future [17, 18, 1]. Example applications requiring such short-lived connections include financial trading through the use of agent technology, interactive bidding, Internet chat rooms, and *etc.*

These limitations coupled with the flexibility and popularity of TCP/IP as a best-effort transport protocol have prompted the network research community to propose and implement a number of techniques that adapt TCP/IP to the Available Bit Rate (ABR) and Unspecified Bit Rate (UBR) services in ATM network environments. This allows these environments to smoothly integrate (and make use of) currently available TCP-based applications and services without much (if any) modifications [16]. However, recent studies [11, 24, 31] have shown that TCP/IP, when implemented over ATM networks, is susceptible to serious performance limitations.

Adapting TCP/IP to ATM: The poor performance of TCP over ATMs is mainly due to *packet fragmentation*. Fragmentation occurs when an IP packet flows into an ATM virtual circuit through the AAL5 (ATM Adaptation Layer 5), which is the emerging, most common AAL for TCP/IP [2] over ATMs. AAL5 acts as an interface between the IP and ATM layers. It is responsible for the task of dividing TCP/IP's large data units (*i.e.*, the TCP/IP packets) into sets of 48-byte data units called *cells*. Since the typical size of a TCP/IP packet is much larger than that of a cell,[1] fragmentation at the AAL is inevitable. In order for a TCP/IP packet to successfully traverse an ATM switching network (or subnetwork), all the cells belonging to that packet must traverse the network *intact*. The loss even of a single cell in any of the network's ATM switches results in the corruption of the entire packet to which that cell belongs. Notice however that when a cell is dropped at a switch, the rest of the cells that belong to the same packet still proceed through the virtual circuit, despite the fact that they are destined to be discarded by the destination's AAL at the time of packet-reassembly, thus resulting in low effective throughput.

There have been a number of attempts to remedy this problem by introducing additional switch-level functionalities to preserve throughput when TCP/IP is employed over ATM. Examples include the Selective Cell Discard (SCD)[2] [3] and the Early Packet Discard (EPD) [31]. In SCD, once a cell c is dropped at a switch, all subsequent cells from the packet to which c belongs are dropped by the switch. In EPD, a more aggressive policy is used, whereby all cells from the packet to which c belongs are dropped, including those still in the switch buffer (*i.e.* preceding cells that were in the switch buffer at the time it was decided to drop c). Notice that both SCD and EPD require modifications to switch-level software. Moreover, these modifications require the switch-level to be aware of IP packet boundaries—a violation of the layering principle that was deemed unavoidable for performance purposes in [31].

The simulation results described in [31] show that both SCD and EPD improve the effective throughput of TCP/IP over ATMs. In particular, it was shown that the effective throughput

[1] This is mainly due to TCP/IP's headers (the minimum number of bytes required for commonly used TCP/IP header fields is 40).

[2] Also called Partial Packet Discard (PPD) in [31].

achievable through the use of EPD approaches that of TCP/IP in the absence of fragmentation. It is important to note that these results were obtained for a network consisting of a single ATM switch. For realistic, multi-hop ATM networks the cumulative wasted bandwidth (as a result of cells discarded through SCD or EPD) may be large, and the impact of the ensuing packet losses on the performance of TCP is likely to be severe. To understand these limitations, it is important to realize that while dropping cells belonging to a packet at a congested switch preserves the bandwidth of that switch, it does not preserve the ABR/UBR bandwidth at all the switches preceding that (congested) switch along the virtual circuit for the TCP connection. Moreover, any cells belonging to a corrupted packet which would have made it out of the congested switch will continue to waste the bandwidth at all the switches following that (congested) switch. Obviously, the more hops separating the TCP/IP source from the TCP/IP destination, the more wasted ABR/UBR bandwidth one would expect even if SCD or EPD techniques are used. This wasted bandwidth translates to low effective throughput, which in turn results in more duplicate data packets transmitted from the source, in effect increasing the response time for the applications.

To summarize, techniques for improving TCP/IP's performance over ATMs based on link-level enhancements do not take advantage of ATM's unique, small-sized cell switching environment; they *cope* with it, and in doing so, these techniques result in a performance that is not appropriate for real-time applications.

Our Research: In a recent study [9], we have unveiled a new transport protocol, TCP Boston, that turns ATM's packet fragmentation problem into an advantage for TCP/IP, thus enhancing the performance of TCP in general and its performance in ATM environments in particular. The rationale that motivates the design of TCP Boston lies in our answer to the following simple question: *Could a partial delivery of a packet be useful?* Our answer is *yes*. In other words, the *en route* loss of one fragment (or more) from a packet does not render the rest of the fragments belonging to that packet useless. TCP Boston manages to make use of such partial information, thus preserving network bandwidth. At the core of TCP Boston is the Adaptive Information Dispersal Algorithm (AIDA), an efficient encoding technique that allows for dynamic redundancy control. AIDA makes TCP/IP's performance less sensitive to cell losses, thus ensuring a graceful degradation of TCP/IP's performance when faced with congested resources. The details of TCP Boston, along with analytical and simulation results that show its superior performance, have been presented in [10].

In this paper, we unveil the real-time capabilities of TCP Boston that allow it to trade off network bandwidth for timeliness through the use of AIDA's Forward Error Correction (FEC) features. We start in section 2 with an overview of TCP Boston. In section 3, we present our

simulation model and experimental setup, and proceed with the presentation of our simulation results, contrasting the performance of TCP Boston to those of TCP Reno and Vegas, and to those of TCP Reno and Vegas with the EPD switch-level enhancements. These simulations demonstrate the superiority of our protocol—measured using both network-centric metrics (*e.g.*, effective throughput and percentage of missed deadlines) and application-centric metrics (*e.g.*, response time and jitter). We conclude in section 4 with a summary and a discussion of our on-going work.

2 TCP Boston: Principles, Protocol, and Implementation

We start this section with an introduction to AIDA. Next, we show how to incorporate AIDA into the TCP/IP stack, and we discuss the various implementation aspects that are incorporated in the current version of TCP Boston (hereinafter interchangeably referred to by "Boston") that is used in our simulations.

2.1 An Introduction to AIDA

AIDA is a novel technique for dynamic bandwidth allocation, which makes use of minimal, controlled redundancy to guarantee timeliness and fault-tolerance up to *any* degree of confidence. AIDA is an elaboration on the Information Dispersal Algorithm of Michael O. Rabin [29], which has been previously shown to be a sound mechanism that considerably improves the performance of I/O systems, parallel/distributed storage devices [5], and real-time broadcast disks [8]. The use of IDA for efficient routing in parallel architectures has also been exploited in [25].

To understand how IDA works, consider a segment S of a data object to be transmitted. Let S consist of m fragments (hereinafter called *cells*). Using IDA's *dispersal operation*, S could be processed to obtain N distinct pieces in such a way that recombining *any* m of these pieces, $m \leq N$, using IDA's *reconstruction operation*, is sufficient to retrieve S. Both the dispersal and reconstruction operations (which can be performed in real-time [6]) are simple linear transformations using *irreducible polynomial arithmetic.*[3]

Several *redundancy-injecting* protocols (similar to IDA) have been suggested in the literature. In most of these protocols, redundancy is injected in the form of parity, which is only used for error detection and/or correction purposes [23]. The IDA approach is radically different in that redundancy is added *uniformly*; there is simply *no* distinction between data and parity. It is this feature that makes it possible to scale the amount of redundancy used in IDA. Indeed, this is the basis for *Adaptive* IDA (AIDA) [7]. Using AIDA, a *bandwidth allocation* operation is inserted after the dispersal operation but *prior* to transmission as shown in figure 1. This bandwidth allocation

[3]For more details, we refer the reader to [29, 8].

step allows the system to *scale* the amount of redundancy used in the transmission. In particular, the number of cells to be transmitted, namely n, is allowed to vary from m (*i.e.*, no redundancy) to N (*i.e.*, maximum redundancy).

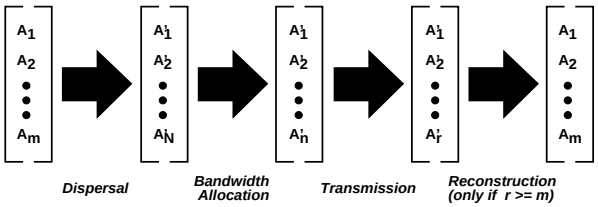

Figure 1: AIDA dispersal and reconstruction

In order to appreciate the advantages that AIDA brings to TCP Boston, we must understand the main difficulty posed by fragmentation. When a cell is lost *en route*, it becomes impossible for the receiver to reconstruct the packet to which that cell belonged unless: (1) there is enough extra (redundant) cells from the packet in question to allow for the recovery of the missing information (*e.g.*, through parity), or (2) the cell is retransmitted. These two alternatives are described next.

Spatial Redundancy: The first solution above suggests the use of spatial redundancy to mask erasures (cell losses). While feasible, such a technique may be quite wasteful of bandwidth (since the redundant information will have to be communicated whether or not erasures occur), and is not likely to help when cell losses exceed the forward erasure capacity of the encoding scheme, which is almost certainly the case since cells are typically dropped in "batches" when switches run out of buffer space. An example of the use of this approach is the study in [12], which suggests the use of *Forward Error Correction* (FEC) for real-time, unreliable video communication over ATM. In that study, FEC was shown to allow the trading of bandwidth for timeliness. FEC's performance was shown to depend on many parameters including the network load, the level of redundancy injected into FEC traffic, and the percentage of connections (traffic) using FEC. FEC was shown to be most effective when corruption is restricted to few cell erasures per data block (*e.g.*, video frame).

Similar to FEC, AIDA supports the use of spatial redundancy to mask erasures. Furthermore, when incorporated with TCP, AIDA allows this support to be fully integrated within the flow control mechanism of TCP, thus making it possible to perform forward error correction *without* necessarily overloading the network resources. For example, if network congestion is detected, one could increase AIDA's level of spatial redundancy (thus protecting against likely cell drops), while decreasing TCP's congestion window size (thus protecting against buffer overflow by reducing the number of bytes "on the wire"). This integration of redundancy control and flow control in a

reliable transport protocol[4] could be quite valuable for real-time communication as reported in [7].

Temporal Redundancy: The second solution above suggests the use of temporal redundancy to recover from erasures. Two possibilities exist—each representing an extreme in terms of the functionality required at the sender and receiver ends. The first extreme would be for the receiver to do nothing, and simply wait for the sender to automatically retransmit all cells from the packet in question as would be dictated by TCP's packet acknowledgment protocol. This is exactly what current adaptations of TCP over ATMs do (including the SCD and EPD techniques). As we explained before, such an approach is not effective in terms of its use of available bandwidth, especially in multi-hop networks. Moreover, these passive techniques are not appropriate for real-time applications as it results in *larger jitter* and thus more uncertainty about communication time. Of course this technique has the advantage of being quite simple to implement since it requires no additional functionality at the sender and receiver ends. The other extreme would be for the receiver to keep track of which cells are missing and then to request retransmission of only those cells. This technique, has the advantage of being effective in terms of its use of available bandwidth, but may result in considerable overhead, especially when the level of fragmentation (*i.e.* number of cells per packet) is high.

The use of AIDA within TCP/IP allows us to reap the advantages of both of the above extremes, while largely avoiding their disadvantages. To explain how this could be done, consider the following scenario. The sender disperses an outgoing m-cell segment (packet) into N cells, but sends a packet of only m of these cells to the receiver, where $N >> m$. Now, assume that the receiver gets r of these cells. If $r = m$, then the receiver could reconstruct the original segment, and acknowledge that it has *completely* received it by informing the sender that it needs *no* more cells from that segment. If $r < m$, then the receiver could acknowledge that it has *partially* received the packet by informing the sender that it needs $(m - r)$ more cells from the original segment. To such an acknowledgment, the sender would respond by sending a packet of $(m - r)$ fresh cells (*i.e.* not sent the first time around) from the original N dispersed cells. The process continues until the receiver receives enough cells (namely m or more) to be able to reconstruct the original segment.

Two important points must be noted. First, using AIDA, *no* additional bandwidth is wasted as a result of cell losses; every cell that makes it through the network is used. Moreover, this cell-preservation behavior is achieved *without* requiring individual cell acknowledgment. Second, using AIDA, *no* modification to the switch-level protocols is necessary. This stands in sharp contrast to the SCD and EPD techniques, which necessitate such a change. The incorporation of AIDA into TCP/IP over ATMs requires only additional functionality at the interface between the IP and

[4]FEC is *not* a reliable transport mechanism.

ATM layers (*i.e.*, the AAL), which we discuss later in the paper.

Adjusting Spatial and Temporal Redundancy for Timeliness: The incorporation of AIDA in TCP allows us to optimize the use of communication bandwidth by *dynamically* balancing the use of spatial redundancy (through FEC) and temporal redundancy (through retransmission) to achieve the level of timeliness desired by the application software.

Figure 2 shows the transmission window managed by AIDA in TCP Boston. As explained before, prior to a packet transmission, AIDA encodes the original m-cell packet into N cells ($N >> m$). Based on network congestion conditions, it dynamically adjusts n the *transmission window size*, which represents the size of the packet to be actually transmitted.

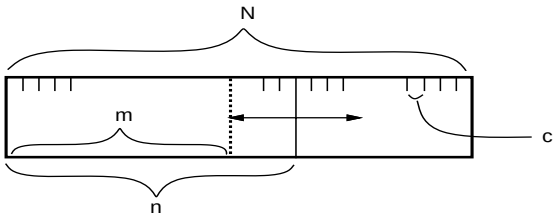

Figure 2: Transmission Window managed by AIDA

The transmission window in TCP Boston can be custom-tuned to meet the spatial redundancy requirements of particular applications or services. For example, time-critical applications may require that the level of spatial redundancy be increased to mask cell erasures (up to a certain level), and thus to avoid retransmission delays should such erasures occur. By avoiding such delays, the likelihood that tight timing constraints will be met is increased (at the expense of wasted bandwidth). In the remainder of this section, we analytically characterize the relationship between spatial redundancy (wasted bandwidth) and temporal redundancy (wasted time). In the next section, we experimentally characterize this relationship.

Relationship Between Spatial Redundancy and Timeliness As we explained before, the loss of a single ATM cell from a TCP packet implies that the delivery of that packet to the IP layer will be delayed by at least one round-trip (until the retransmitted packet is received). This delay could be minimized if Boston's spatial redundancy feature is used. In this section, we quantify the relationship between timeliness and spatial redundancy—*i.e.*, the relationship between wasted bandwidth and response time.

In our analysis, we assume that the network has a single congested switch, whereby $(1-p)$ is the probability that a cell will be dropped at that switch. We assume a memory-less switch behavior, which means that the dropping of a cell at a switch is an event *independent* of the feat of previous

cells transmitted through that switch. To simplify the following derivations, we will assume that a constant number of redundant cells is always added to a packet,[5] independent of the packet size. Let α denote the number of extra cells added by TCP Boston per packet. Thus for every m-cell packet, Boston sends α redundant cells for a total of $m + \alpha$ cells. For example, if $m = 10$ and $\alpha = 1$, then for every 10-cell packet, TCP Boston sends 11 cells, of which *any* 10 cells are sufficient to reconstruct the original packet.

Under these assumptions, packet retransmission would be necessary only when more than α cells are lost *en route*. Thus, the probability of a retransmission is equal to the probability that in a sequence of $m+\alpha$ Bernoulli trials (to send cells through the congested switch), more than α cells are lost. Let W denote that probability.

$$\begin{aligned} W &= \sum_{i=\alpha+1}^{m+\alpha} \left(\begin{array}{c} m+\alpha \\ i \end{array} \right) (1-p)^i \cdot p^{m+\alpha-i} \\ &= 1 - \sum_{i=0}^{\alpha} \left(\begin{array}{c} m+\alpha \\ i \end{array} \right) (1-p)^i \cdot p^{(m+\alpha)-i} \end{aligned} \tag{1}$$

To simplify the following derivations, we assume that the probability of further retransmissions (*i.e.* beyond the first retransmission) is also W. Notice that this is a fairly conservative assumption for TCP Boston since a retransmission will involve much less than the original $m + \alpha$ cells, and the probability of losing α of these fewer number of cells will be *considerably* smaller.

Let U denote the average number of *"round trips"* necessary to deliver a packet. From equation 1, the probability of needing *one* round trip would be $(1 - W)$ (*i.e.* succeeding in the first round); the probability of needing *two* round trips would be $W(1 - W)$ (*i.e.* failing the first round and succeeding in the second round); ... *etc.* In general, the probability of needing j round trips to deliver a packet would be $W^{j-1}(1 - W)$, which leads to the following expression for U.

$$\begin{aligned} U &= (1-W) \cdot \sum_{j=1}^{\infty} j \cdot W^{j-1} \\ &= (1-W) \cdot \frac{1}{(1-W)^2} \\ &= \frac{1}{(1-W)} \end{aligned} \tag{2}$$

[5] A more realistic scenario would assume that the number of extra redundant cells is a percentage of the total number of cells per packet.

Substituting from equation 1, we get

$$U = \frac{1}{\sum_{i=0}^{\alpha} \left(\begin{array}{c} m+\alpha \\ i \end{array} \right) (1-p)^i p^{m+\alpha-i}} \quad (3)$$

Obviously, U is a measure of response time (in terms of the number of round trips), $(1-p)$ is a measure of switch congestion (*i.e.* p is a measure of transmission reliability), and $S = \frac{\alpha}{m}$ is a measure of spatial redundancy.

The above relationships could be used to set the desired level of spatial redundancy in TCP Boston. For example, if the network Maximum Transfer Unit (MTU) (*i.e.* packet size) is 1.5 kB, thus resulting in a fragmentation at the ATM switch of $m = 30$, then figure 3 shows the relationship between the timeliness (on the Y axis) and spatial redundancy (on the X axis) as predicted using equation 3 for various cell loss rates. Using figure 3, and in order to ensure that on the average packet transmissions will be completed within 3 round trip times in an environment where the worst-case cell loss rates are known[6] to be less than 10%, the appropriate number of extra cells (α) to be used should be 3, making the necessary percentage of bandwidth to trade off for timeliness equal to 10%. If the worst-case cell loss rates are known to be less than 5%, then the appropriate value for α would be 1, making the necessary percentage of bandwidth to trade off for timeliness equal to 3%.

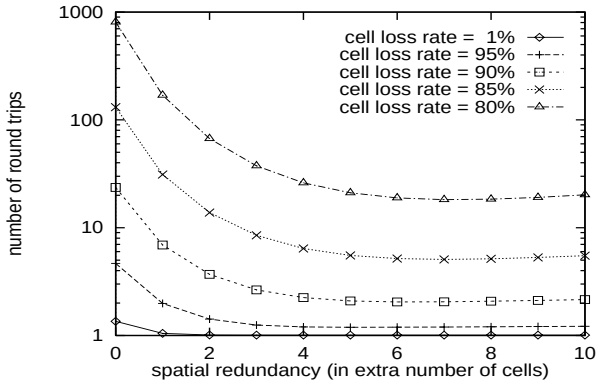

Figure 3: Response time Prediction: The average number of round trips is plotted on the y-axis (log scale) as a function of α, the number of cells to be added to a 30-cell (*i.e.*, 1.5-kB) packet, for cell loss rates of 1, 5, 10, 15, and 20 percent.

[6]This is possible if such a rate is advertised for a particular class of service, or if the cell loss rates for a connection could be measured dynamically.

2.2 Overview of TCP Boston

In this section we explain the essential aspects of our implementation of TCP Boston, with a special emphasis on those elements that are uncommon in other TCP implementations.

The purpose of this protocol is to provide a reliable transfer of data for end-to-end applications. The protocol, when properly tuned, can be implemented over both ATM and packet-switched networks. But, since it is designed in such a way that it takes advantage of ATM's relatively small-sized cell (*i.e.*, 53 bytes) environment, it can achieve a high performance gain when it is deployed over ATM networks.

The main functions included in the protocol are: *session management*, *segment management,* and *flow control and transmission.* We give a summary of these functions below:

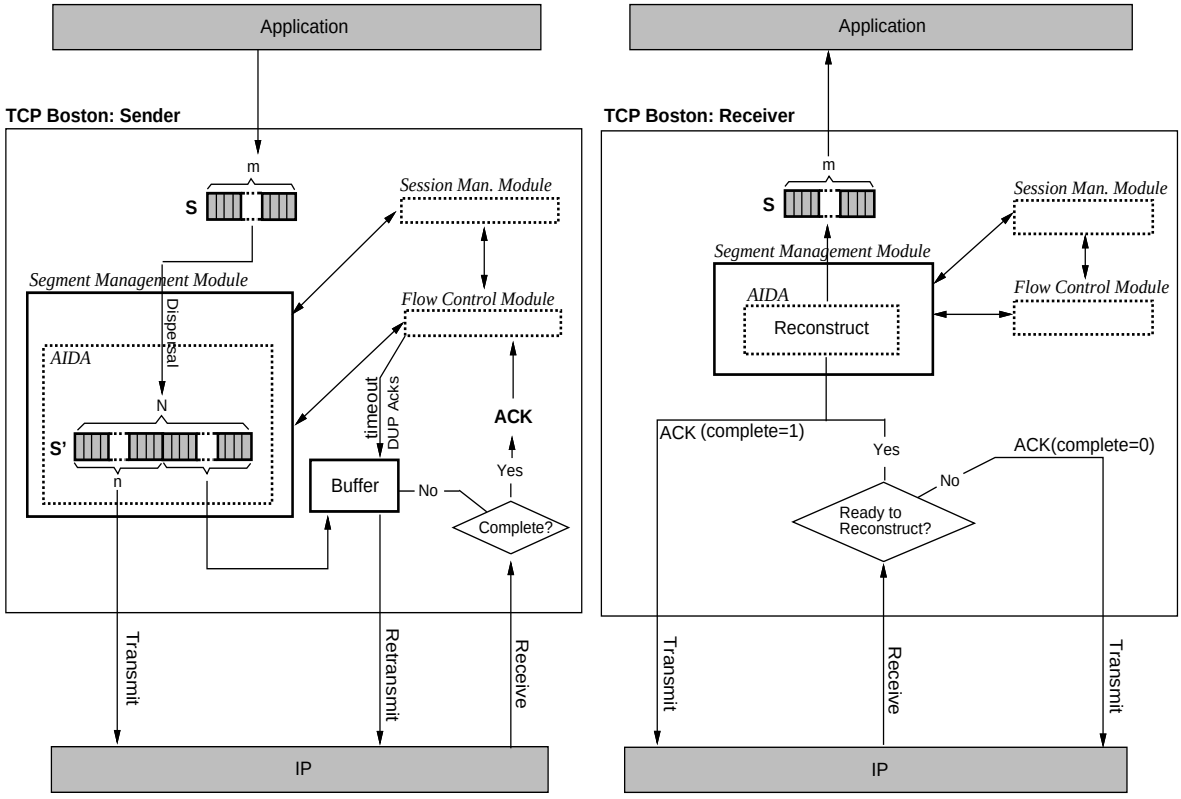

Figure 4: TCP Boston: Outline of Protocol Implementation

Session Management: The protocol manages a TCP session in three phases: a *connection establishment* phase, a *data transfer* phase, and a *termination* phase. The purpose of these phases, as well as the functions performed therein, generally follow those of current TCP implementations, except that information specific to IDA which are required by the receiver for reconstruction pur-

poses (such as the value of m for example), are piggy-backed onto the protocol packets during the three-way handshaking at the connection establishment phase.[7]

Segment Management: Processes for (1) segment encoding (at the source) and (2) segment reconstruction (at the sink) are unique to TCP Boston. These processes are described below.

- *Segment encoding:* Given a data block (segment) of size b bytes, the protocol divides the data block into m cells of size c, where $m = b/c$ bytes. Next, the m cells are processed using IDA to yield N cells for some $N >> m$. For example, if $b = 1{,}000$ bytes and $c = 50$, then $m = 20$, and N could be set to 40. For each cell, one byte of heading is required for identification purposes. This would be needed during reconstruction at the receiver end. Once this encoding is done, the first m cells from the segment are transmitted as a single packet and the unused $N - m$ cells are kept in a buffer area for use when (if) more cells from that segment must be transmitted to compensate for lost cells (see below).

- *Segment reconstruction*: When a packet of cells is received, the protocol first checks if it has accumulated m (or more) different cells from the segment that corresponds to that packet. If it did, it reconstructs the original segment using the proper IDA reconstruction matrix transformation and signals the flow control component to send an acknowledgment (hereinafter referred to as an `ACK`) indicating that reconstruction was successful. If not, it keeps the received cells for later reconstruction, and signals the flow control component to send an `ACK`, piggy-backed with the number of cells that have been accumulated so far from the segment. Such an `ACK` would inform the sender that reconstruction is not possible, and that the pending number of cells from that segment need to be transmitted at the time of next packet retransmission.

Flow Control and Transmission: Flow control determines the dynamics of packet flow in the network, which in turn affects the end-to-end performance of the system. Any feedback-based TCP flow control algorithm (*e.g.*, Tahoe, Reno, and Vegas) can be used with TCP Boston with a minor modification to handle the revised feedback mechanism of TCP Boston. When an `ACK` arrives, the sender checks a flag to determine if that `ACK` signals the successful reconstruction (at the receiver) of a segment. If it does, the sender calls the standard `ACK` procedure. If it doesn't, the sender extracts from the `ACK` the number of cells r received so far (see above) and then prepares $m - r$ additional cells from the desired segment in a single *new* packet that will be transmitted at the next retransmission time. This process continues until the receipt of an `ACK` from the receiver indicating

[7] For efficiency, such information could be permanently *"coded"* into TCP Boston.

that the segment has been successfully reconstructed, in which case any remaining cells from that segment are discarded from the sender's buffer.

Notice that the partial delivery of a packet does not result in updating the received-segment number for the receiver's TCP window manager.[8] Also, an `ACK` signaling a partial packet delivery does not cause an increase in the sender's congestion window. Rather, it acts as a hint to the sender to update the number of cells included in the next packet retransmission.

In our current implementation, the protocol is composed of three main modules: a *Session Management Module*, a *Segment Management Module*, and a *Flow Control Module*. Each of these modules executes the corresponding function described in the previous section. Figure 4 depicts the configuration and interaction of the three modules for both the sender and the receiver. Due to space limitations, we have omitted the details of these modules. The interested reader is referred to [9].

3 Performance Evaluation

In this section we analyze the performance of TCP Boston, based on two different strategies. The first strategy is *bandwidth preserving*, and thus does not employ any redundancy (*i.e. no* FEC). The second strategy trades bandwidth for timeliness by allowing the use of FEC speculatively. We start with a description of our simulation environment. We follow that with an evaluation of the performance of TCP Boston under both bandwidth preserving and FEC environments (using Vegas-style flow control), and compare it with that of TCP Vegas. In particular, we concentrate on real-time performance metrics—namely response time, the percentage of missed deadlines and the variability of packet communication time (as a measure of jitter).

In the remainder of this paper, we use "Boston-BP" (or simply "Boston") to refer to the Bandwidth Preserving TCP Boston protocol, and we use "Boston-FEC" to refer to the TCP Boston with FEC protocol.

3.1 Simulation Environment

We measure the performance characteristics TCP Boston and TCP Reno (or Vegas) under UBR service in ATM networks. Figure 5 illustrates the network topology used in the simulation.

The simulated network consists of 16 source nodes and 1 sink node, where all the nodes are connected to a single switch. Each link bandwidth is set to 1.5 Mbps with propagation delay of 10 msec. The link bandwidth does not represent any particular technology; it was chosen to simulate a relatively low bandwidth-delay product (approximately 74 cells) network, without adjusting the

[8] This enables the receiver to send duplicate `ACK`s to signal a packet drop to the sender.

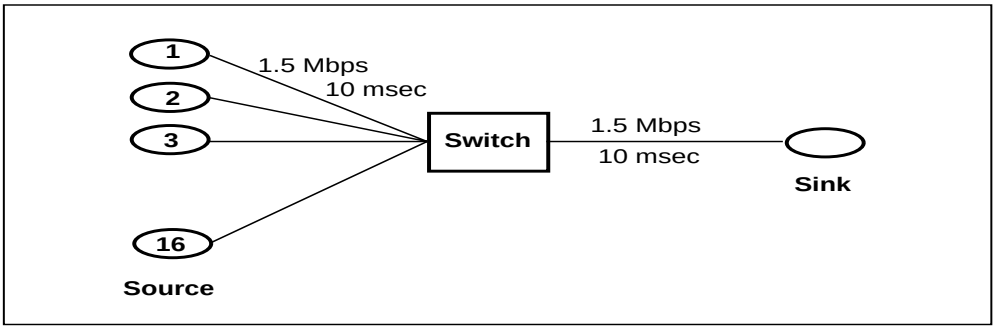

Figure 5: Configuration of simulated network.

TCP performance parameters that are sensitive to link delays. This configuration simulates a WAN environment with a radius of 3,000 km and a bottleneck link bandwidth of 1.5 Mbps.

Our simulations use a total of 16 TCP connections, each is established for one of the configuration's source-sink pairs. Each source generates an infinite stream of data bytes. Each simulation runs for 700 simulated seconds to transfer a total of 120 MB of data.

ATM Switching Model: The ATM switch is a simple, 16-port output-buffered single-stage switch [15]. When the output port is busy, a cell at the input port is queued into the output buffer of the simulated switch. When the output-buffer is full, an incoming cell destined to the output port is dropped. The output buffer is managed using FIFO scheduling, and cells in input ports are served in a round-robin fashion to ensure fairness.

In our simulations, the ATM Adaptation Layer (AAL) implements the basic functions found in AAL5, namely fragmentation and reconstruction of IP packets [2, 20]. AAL divides IP packets into 48-byte units for transmission as ATM cells, and appends 0 to 47 bytes of padding to the end of data. To support TCP Boston the destination AAL reconstructs a packet out of the received cells even when the resulting packet is incomplete. Incomplete packets are discarded by the destination AAL for Reno and Vegas implementation.

Baseline Parameters: The parameters used in the simulation include the TCP packet size, the TCP window size, and the switch buffer size. Three different packet sizes were selected to reflect maximum transfer unit (MTU) of popular standards: 576 bytes for IP packets, 1,500 bytes for Ethernet, 4,470 bytes for FDDI link standards [26], and 9,180 bytes which is the recommended packet size for IP over ATM [4]. The values for the TCP window size are 8 kB, 16 kB, 32 kB, and 64 kB. Buffer sizes used for the ATM switch are 64, 256, 512, 1,000, 2,000, and 4,000 cells.

To measure the effectiveness of Boston-FEC's dynamic redundancy features, we used a TCP Vegas module as the base flow control mechanism. Unlike its counterparts (*i.e.*, Tahoe and Reno),

Vegas employes a proactive flow control mechanism, which provides better congestion forecast by detecting the incipient stages of congestion before losses start to accrue (rather than using the loss of segments as a signal of congestion)[13]. It provides measured and predicted network bandwidth usage for a TCP connection, and measured round trip time (RTT) of a packet, along with some threshold values that are used to manage the flow control effectively. To minimize the spatial overhead, Boston's redundancy control mechanism requires accurate congestion forecast, so that it can dynamically adjust the rate of spatial redundancy according to the current network load, and in this regard, Vegas can provide useful information for the effective management of Boston's dynamic redundancy control scheme.

In our experiments for Boston-FEC we fixed the amount of redundancy injected into the data as follows: for normal (initial) data transmissions, a fixed 3.3 percent of redundancy is injected, and for retransmitted data packets, 10 percent of redundancy is added to its original segment size. In addition, segments that belong to current transmission window are multiplexed, so that the cells that belong to a packet are interleaved in the transmission line. This multiplexing is beneficial since cell (or packet) drops are likely to occur in "bundles" [12], and multiplexing indeed improves the performance in FEC method by spreading the cell erasures rather evenly over multiple packets, resulting in an increased success rate in fewer rounds of retransmissions for each packet. In our simulation experiments, multiplexing provided up to 3.6 percent decrease in response time, depending on the packet size and other parameters, such as switch buffer and congestion window size.

Simulation Engine: The LBNL Network Simulator (ns) [19] was used for both packet-switched and ATM network simulations. To simulate TCP Boston, we modified ns extensively to implement the three main modules (*i.e.*, the *Session Management*, *Segment Management*, and *Flow Control* modules) described in the previous section. Since ns is originally designed to support packet-switched network environments, major modifications were necessary to allow it to support ATM-like network environments. In particular, the essential functions of AAL5 were added to simulate the handling of IP packets (*i.e.*, fragmentation and reassembly of IP packets) [2, 20]. Also, the link layer of ns has been modified to include basic functions of ATM switches and virtual circuit management. To enable us to exploit Boston's FEC features, we have also added a TCP Vegas module to complement the existing Reno module of ns. In addition to the above necessary modifications, the ns package has also been enhanced to allow for the gathering of additional performance statistics, such as *effective throughput* (hereinafter interchangeably termed *goodput*), *cell loss rate*, *effective packet loss rate*, *response time*, *missed deadlines*, and jitter.

3.2 Performance Characteristics of TCP Boston

For a baseline performance characterization of Boston-FEC, Boston-BP, and Vegas we rely on three metrics: effective throughput, response time, and packet loss rate. The results obtained for these metrics are similar to those presented in [10], where the performance of a Reno-based implementation of TCP Boston (as opposed to the Vegas-based implementation presented here) was characterized.

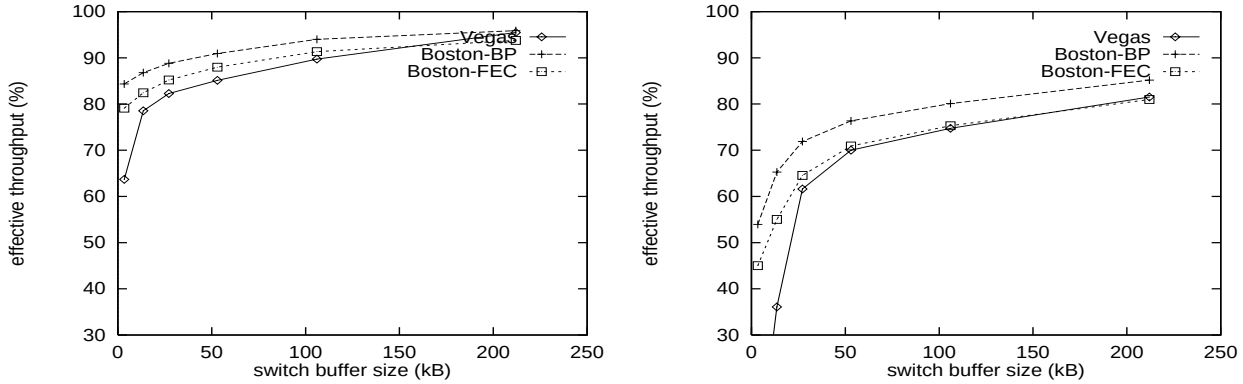

Figure 6: Comparison of the goodput of Vegas, Boston-BP, and Boston-FEC over ATM, for 1,500-byte packets (left) and 9,180-byte packets (right) and a 64 kB window size, as a function of switch buffer size

Effective Throughput Characteristics: The effective throughput (or goodput) refers to the throughput where only the bytes that are useful at the application layer are considered. Figure 6 gives a comparison of the goodput achieved by: Vegas, Boston-BP, and Boston-FEC, under 64 kB window size for packet size of 1,500 bytes (left) and 9,180 bytes (right), where the effective throughput of the three different methods is plotted as a function of switch buffer size. Our results show that Boston-BP outperforms both Boston-FEC and Vegas throughout the range of switch buffer sizes, for both small (1,500-byte) and large (9,180-byte) packets. Boston-FEC outperforms Vegas for small and medium buffer sizes. This advantage vanishes as the buffer size increases, and eventually (for the largest buffer size measurement of 212 kB) Vegas edges ahead of Boston-FEC. For both packet sizes, the effective throughput of Boston-BP is greater than that of Boston-FEC over the entire range of switch buffer sizes, with the performance gap getting larger as the buffer size gets smaller.

The effective throughput is closely related to the amount of redundant data transmitted over the network. Such redundancy may be the result of retransmissions due to cell losses and/or the result of spatial redundancy used by the transport protocol for FEC, such as the redundancy injection

found in Boston-FEC. Compared to Boston-BP, the decreased effective throughput of Vegas stems mainly from retransmissions, whereas the decreased effective throughput of Boston-FEC stems mainly from its use of spatial redundancy for FEC.

Response Time Characteristics: We define the *packet response time* as the elapsed time from the transmission of the first byte until the receipt of the last byte (including all the necessary dispersal and retrieval processing). Also, we define the *average packet response time* (or simply *response time*) to be average response time for all packets in a single TCP connection:

$$\text{Response Time} = \frac{\sum_{i=1}^{N} [Recv\_Time(i) - Send\_Time(i)]}{N}$$

where,

$Recv\_Time(i)$ = the time the last byte for packet i is received by a receiver
$Send\_Time(i)$ = the time the first byte for packet i is transmitted by a sender
N = total number of data packets received during a TCP connection

Figure 7 shows the response time of the three methods for the packet size of 1,500 bytes (left) and 9,180 bytes (right) under 64 kB TCP window size, where the average response time of the three different methods is plotted as a function of switch buffer size. Figure 8 is the same as Figure 7, except that its range for the x and y axes have been reduced to enlarge the region where data points are closely clustered. In general, as the switch buffer size becomes larger, the response time increases accordingly, due to the fact that the average number of cells queued at each switch increases as the switch buffer size becomes large. This phenomenon is well depicted in the two response time plots (and has been documented in other studies [27] as well). The plots also imply that the queuing delay increases linearly as the buffer size increases.

Boston-FEC outperforms Boston-BP (not to mention Vegas) in response time characteristics over the entire buffer range. Moreover, the relative gap (*i.e.*, ratio) between Boston-FEC's performance and that of Boston-BP increases as the buffer size shrinks. In the plot, though the gap of response times for the three methods seems to increase as the buffer size increases, the actual ratio of response times—which is a more accurate measure for comparison purposes—increases as the buffer size becomes smaller. For instance, our measurements show that the ratio of response times for the three methods using a small buffer size (3.4 kB) is, Vegas:Boston-BP:Boston-FEC = 1 : 0.96 : 0.90, whereas the ratio using a large buffer size (212 kB) is 1 : 0.97 : 0.96. This means that, when network resources—such as switch buffers—become scarce, the cell loss rate at the switch tends to increase. As the cell loss rate increases, the advantage of Boston-FEC's redundant data

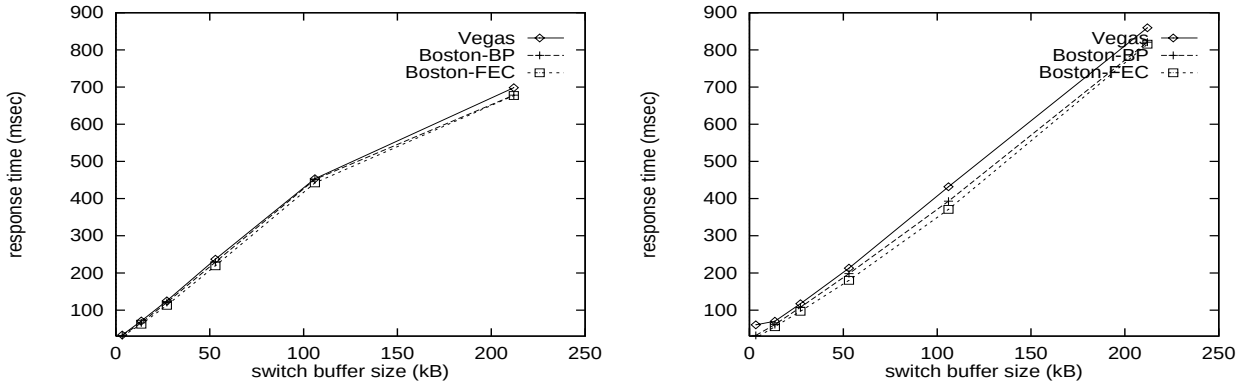

Figure 7: Comparison of response times for Vegas, Boston-BP, and Boston-FEC over ATM, for 1,500-byte packets (left) and 9,180-byte packets (right) and a 64 kB window size, as a function of switch buffer size

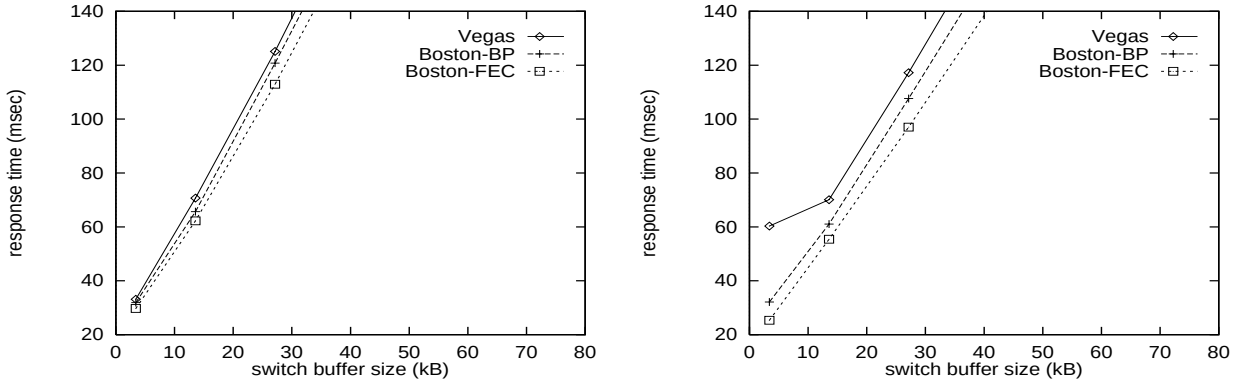

Figure 8: Partial (enlarged) view of response times for Vegas, Boston-BP, and Boston-FEC over ATM, for 1,500-byte packets (left) and 9,180-byte packets (right) and a 64 kB window size, as a function of switch buffer size

transmission materializes, by increasing the probability that clients complete packet reception in a lesser number of retransmissions.

Packet Loss Rates: The packet loss rate (or simply loss rate) refers to the percentage of packets lost due to cell drops at the ATM switch. Figure 9 compares the loss rates of the three protocols for packet sizes equal to 1,500 bytes (left) and 9,180 bytes (right), where the loss rate is plotted as a function of switch buffer size. Both Boston-FEC and Boston-BP show a big performance edge over Vegas. The loss rates of Boston-FEC and Boston-BP are almost identical, with Boston-BP's loss rate consistently less than that of Boston-FEC. The average difference is a mere 1 percent for 1,500-byte packets, and 2 percent for 9,180-byte packets. The slight increase in loss rate for Boston-FEC over Boston-BP is mainly due to the non-bandwidth-preserving nature of Boston-FEC, which results in an increased amount of traffic due to the injection of redundancy.

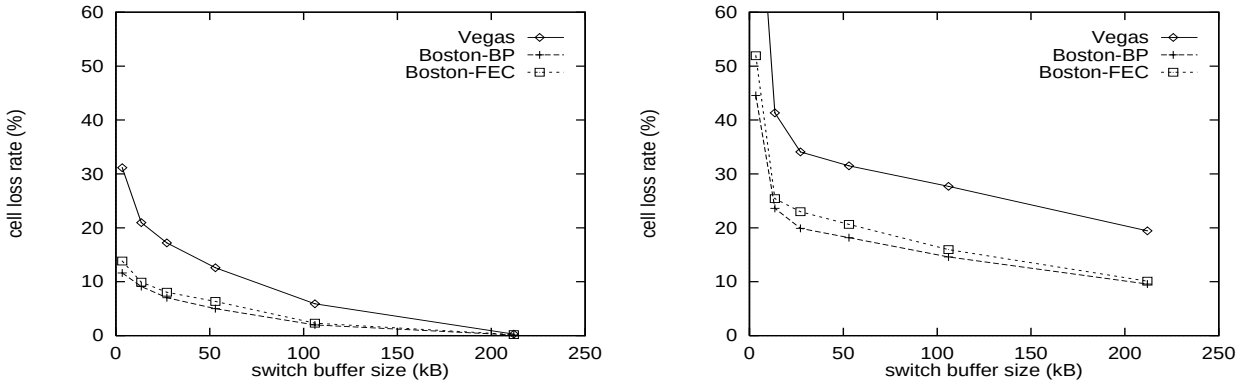

Figure 9: Comparison of packet loss rates for Vegas, Boston-BP, and Boston-FEC over ATM, for 1,500-byte packets (left) and 9,180-byte packets (right) and a 64 kB window size, as a function of switch buffer size

Effect of Window Size: So far, the results we have presented for Boston-BP, Boston-FEC, and Vegas were under a TCP window size equal to 64 kB. The results for the three protocols under window sizes of 32 kB, 16 kB, and 8 kB show a gradual convergence in performance as the window size decreases. These results are not shown here for space limitations.

Effect of TCP Boston on Flow Control: Boston's ability to accept incomplete packets (as opposed to counting such packets as lost ones) is likely to impact the flow control behavior by making it less sensitive to network congestion, and thus more aggressive in its use of network bandwidth. To understand how this could happen, it suffices to note that using Boston, retransmitted packets are smaller (containing only the pending number of cells) and thus more likely to be delivered intact.

Therefore, the likelihood that a sender utilizing TCP Boston will detect a packet loss (as a result of repeated acknowledgments received for the same packet) is reduced, which in turn, increases the probability that the sender will not decrease the congestion window (not to mention the possibility that it may even increase it). The result of this phenomenon is a minute increase in Boston's cell loss rate, compared to the actual cell loss rate of Reno (or Vegas). In our experiments presented in [9], Boston exhibited a maximum of 2% higher cell loss rate than the actual cell loss rate of Reno (or Vegas). This percentage becomes smaller as the cell loss rate increases.

Despite Boston's aggressive use of bandwidth, it conserves the basic dynamics of the underlying TCP flow control, without causing adverse effects on the traffic flow in the network. Instead, it brings an increased effective throughput, which in turn results in an overall increase in other performance categories, such as response time, retransmission rate, and cell loss rate [9].

3.3 Real-time Features of TCP Boston

In this section we establish the suitability of TCP Boston for real-time applications by showing the effect of Boston's FEC on the percentage of missed deadlines and on jitter (variability in response time between successive packets).

Percentage of Missed Deadlines: When retransmissions occur due to cell losses, a client (*i.e.*, receiver) may require longer wait-time to receive a packet. When network resources become limited, this phenomenon tend to be severe, and as a result, a client may have to wait for multiple RTTs to receive a single packet. For applications that require timely reception of data, packets received after a preset delay become useless. Such a preset delay could be considered as a *firm deadline* on packet communication time.

We measured the percent of packets that miss such a deadline, and compared the results for the three protocols. We fixed the dealine for packet delivery at 700 msec for 1,500-byte packets and at 1,400 msec for 9,180-byte packets.[9] The results of these experiments are shown in figure 10, where the percentage of packets that miss their "firm" deadline is plotted on the y-axis as a function of buffer size. Figure 11 is an enlarged view of figure 10, emphasizing the medium-sized buffer range. For 1,500-byte packets, Boston-BP showed an average of 18.9 percent improvement over Vegas, and an overwhelming average of 41 percent for 9,180-byte packets. Boston-FEC showed an average of 19.2 percent improvement over Vegas for 1,500-byte packets, and a 42 percent for 9,180-byte packets. Boston-FEC showed only a mild improvement over Boston-BP, with an average of about 1.5 percent for 1,500-byte packets and 2.2 percent for 9,180-byte packets.

[9]The two deadline values (*i.e.*, 700 and 1,400 msec) have been selected to acquire reasonable rates of deadline-missing packets throughout the entire buffer ranges for both packets sizes presented in the plot.

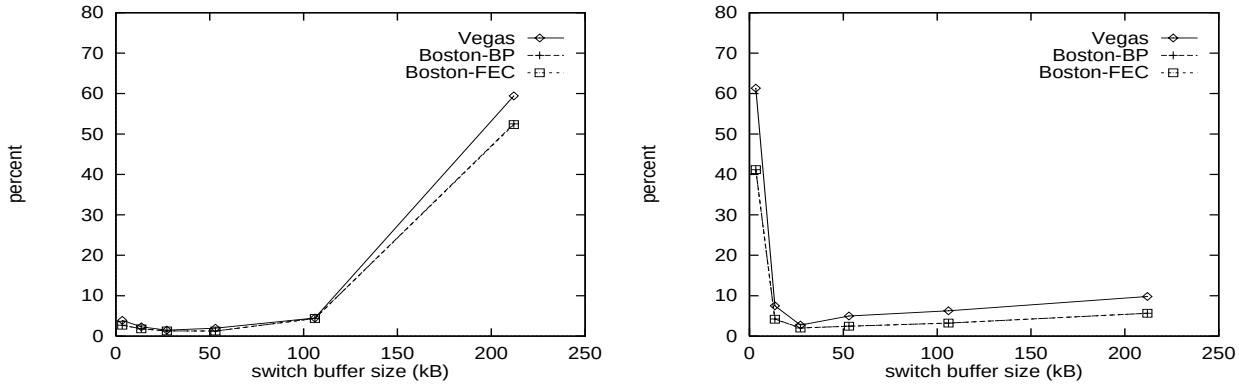

Figure 10: Percent of missed deadlines for Vegas, Boston-BP, and Boston-FEC over ATM, for a packet size of 1,500 byte (left) and 9,180 byte (right) and 64 kB window size, as a function of switch buffer size

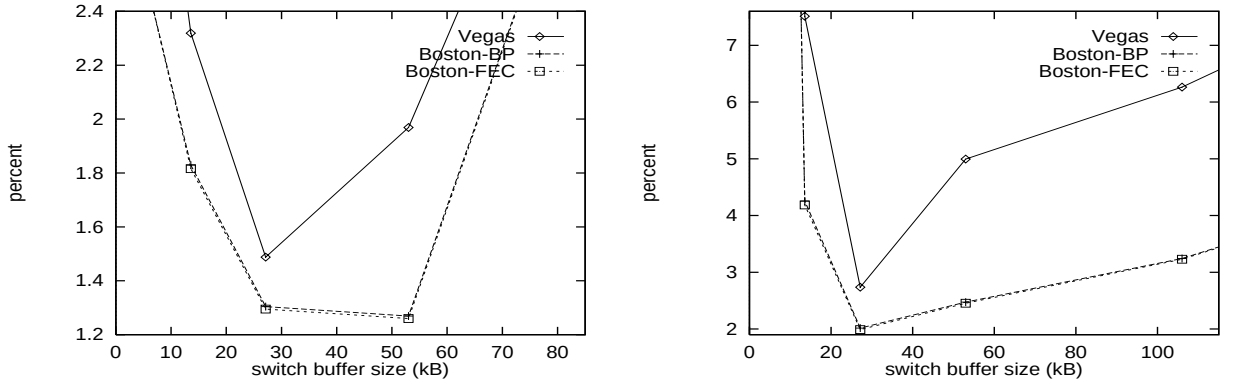

Figure 11: Partial (enlarged) view of percent of missed deadlines for Vegas, Boston-BP, and Boston-FEC over ATM, for a packet size of 1,500 byte (left) and 9,180 byte (right) and 64 kB window size, as a function of switch buffer size

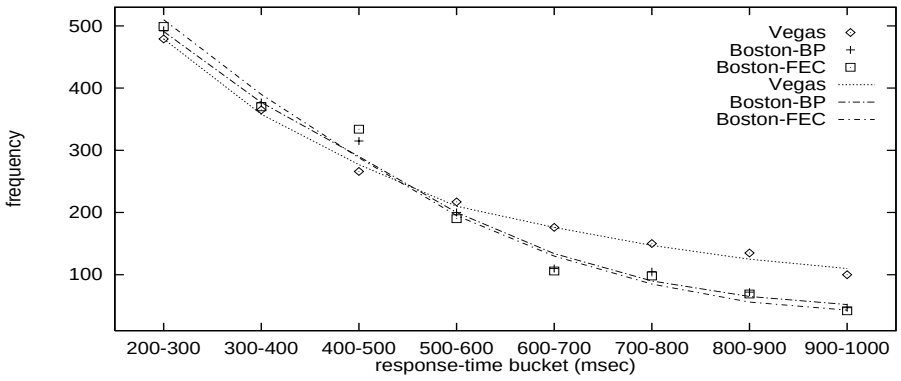

Figure 12: Histogram of packet response time for Vegas, Boston-BP, and Boston-FEC under 64 kB TCP window, 13.6 kB switch buffer, and 1,500 byte packet size. Response times are grouped in 100-msec intervals. The '0-100', '100-200', and 'over 1,000' msec intervals are omitted. Measurements are shown as points and trends are shown as lines.

Response Time Variability: For many real-time applications, the variability in the response time for successive transmissions (or jitter) must be bounded to ensure that allocated buffers will be able to "smooth out" the jitter in the real-time data stream. Figure 12 depicts the distribution (shown as a histogram) of response times observed in a typical simulation. It shows that the distribution of the response times for Vegas exhibits a larger variation than that of Boston-BP, which in turn exhibits a larger variation than that of Boston-FEC. One way of measuring such variability is by computing the standard deviation of the response time distribution. For real-time applications, a small standard deviation is desirable as it would indicate a more "predictable" communication network.

Figure 13 shows the standard deviation of the response time as a function of switch buffer size, under 64 kB TCP window size, for packet sizes of 1,500 bytes (left) and 9,180 bytes (right). In both cases, Boston-BP and Boston-FEC show smaller standard deviations—and thus reduced variability in response time—compared to Vegas.

The standard deviation in response time increases for extremely small buffer sizes. This is due to the increased cell loss rates (more so for Vegas than for Boston-BP and Boston-FEC) as shown in figure 9. The higher cell loss rate means that the probability of repeated packet retransmissions becomes high, resulting in an increased likelihood of very long response times, which in turn results in a higher standard deviation as shown for smaller buffer sizes in figure 13. Obviously, the impact of cell losses on Vegas are more pronounced due to its intolerance to fragmentation relative to Boston-BP and Boston-FEC. The variability in response time increases as well for extremely large buffer sizes. This is due to the longer response times that are possible due to queuing at the switch

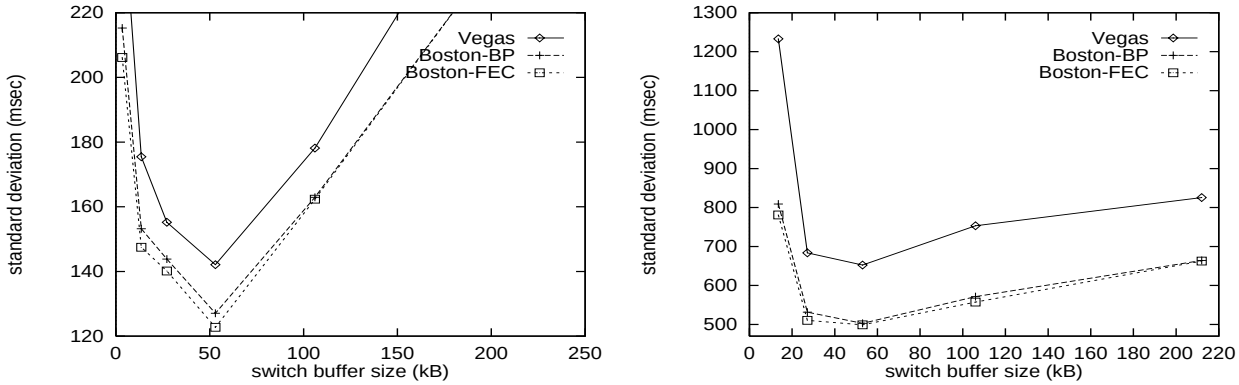

Figure 13: Standard deviation of response time for Vegas, Boston-BP, and Boston-FEC over ATM, for the packet size of 1,500 byte (left) and 9,180 byte (right) and 64 kB window size, as a function of switch buffer size

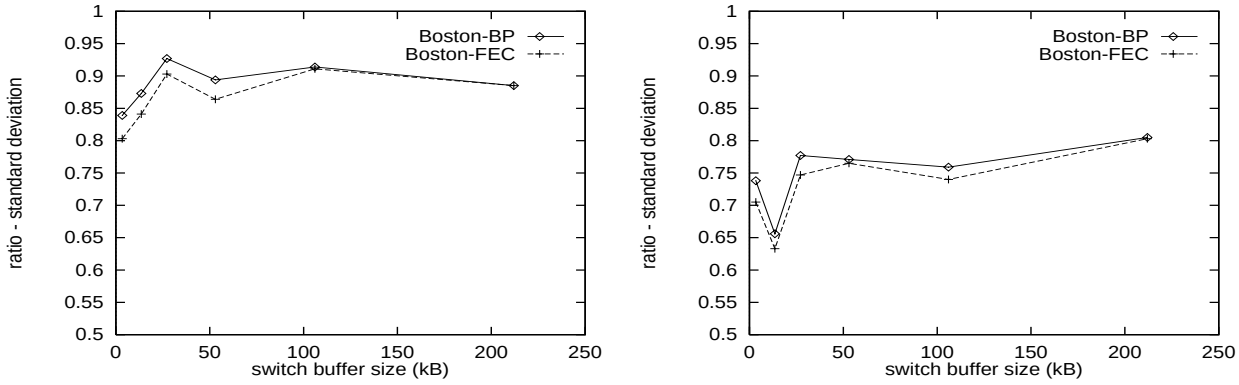

Figure 14: Ratio of the standard deviation for Boston-BP (Boston-FEC) relative to that of Vegas as a function of switch buffer size, for a packet size of 1,500 byte (left) and 9,180 byte (right) and a 64 kB window size

buffers as explained for the results illustrated in figure 7.

Figure 14 shows the ratio between the standard deviation of Boston-BP and that of Vegas, as well as the ratio between the standard deviation of Boston-FEC and that of Vegas for various switch buffer sizes.

In our experiments, Boston-BP showed an average of 32 percent lower standard deviation than that of Vegas under large packet size. Under small packet size, the average standard deviation for Boston-BP showed an improvement of 21 percent over Vegas. For both small and large packet sizes, Boston-FEC showed an additional improvement over Boston-BP. The improvement is more pronounced for small and medium-size buffers, which demonstrates the benefits (in terms of reducing jitter) of using Boston-FEC for a real-time application when network resources are scarce.

Since the response time (and thus the standard deviation of the response time) varies widely for various switch buffer sizes, a more consistent measure of variability—and thus (un)predictability–would be one that normalizes the standard deviation of the response time relative to its mean (using the Z-score theory [14]). Figures 15 and 16 show the behavior of such a *variability index*. It indicates that Boston-BP's variability index is 25 percent lower than that of Vegas for large packets, and 13 percent lower for small packets.

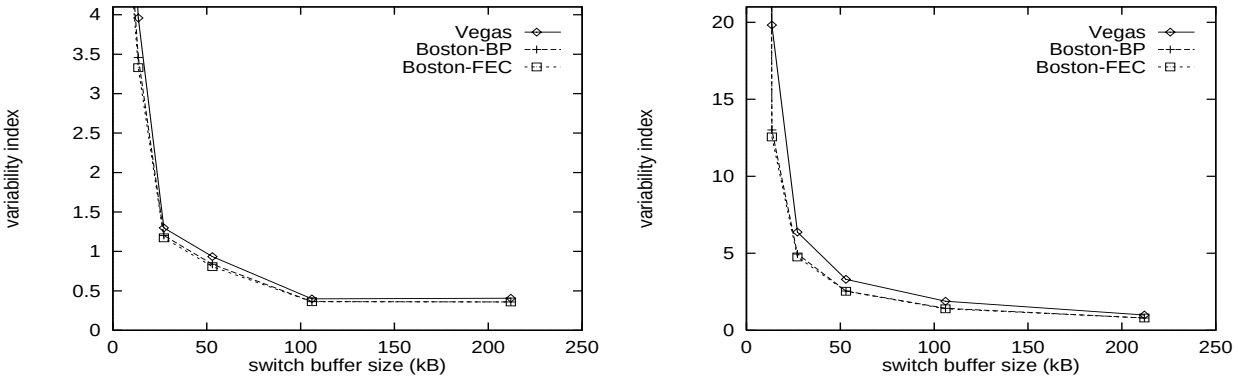

Figure 15: Variability index of response time for Vegas, Boston-BP, and Boston-FEC over ATM, for the packet size of 1,500 bytes (left) and 9,180 bytes (right) and 64 kB window size, as a function of switch buffer size

Summary of Performance Evaluation: Both Boston-FEC and Boston-BP greatly outperform Vegas in all measured performance categories, except under very large switch buffer sizes, where Vegas slightly surpasses Boston-FEC with respect to effective throughput. In this section, Boston-FEC and Boston-BP have been used as representatives for two strategies: Boston-FEC's strategy is to efficiently trade off network bandwidth to improve timeliness—a strategy suitable for applications

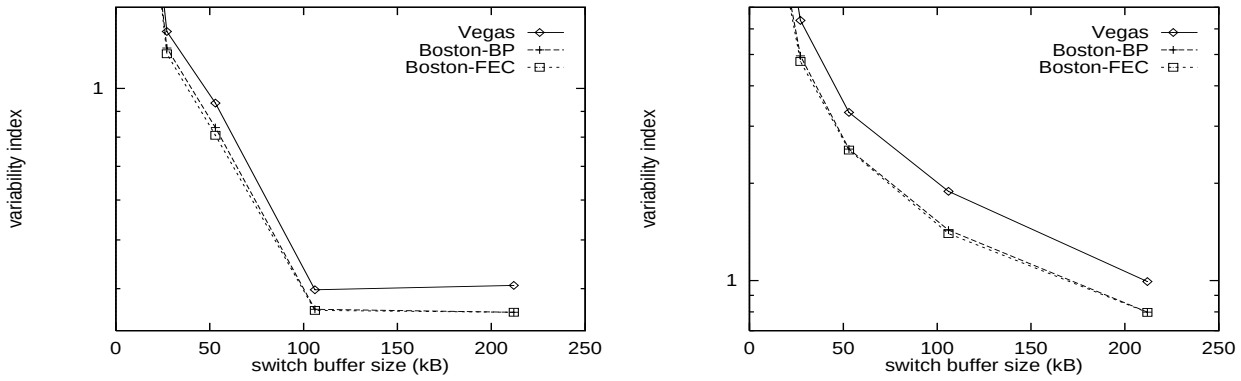

Figure 16: Variability index (on a log scale) of response time for Vegas, Boston-BP, and Boston-FEC over ATM, for the packet size of 1,500 byte (left) and 9,180 byte (right) and 64 kB window size, as a function of switch buffer size

with real-time constraints on transmission delays. Boston-BP's strategy is to preserve network bandwidth—a strategy suitable for non-real-time applications. *Both* strategies are integrated in a *single* protocol—TCP Boston—making it possible to seamlessly support applications with various timeliness requirements.

4 Conclusion

In this paper we exposed the characteristics of TCP Boston that makes it especially suited for real-time applications. TCP Boston integrates a standard TCP/IP protocol (such as Reno or Vegas) with a powerful encoding mechanism based on AIDA (an adaptive version of Rabin's IDA dispersal and reconstruction algorithms [29]). In that respect, we have shown the performance superiority of TCP Boston when compared to plain TCP techniques that are more vulnerable to fragmentation, namely TCP Reno (or Vegas) and TCP Reno (or Vegas) with EPD switch-level enhancements. Our characterization of TCP Boston was two-pronged, using both analysis and simulation. First, we derived the relationship between spatial and temporal redundancy and showed how such a relationship could be used to select the appropriate level of communication bandwidth to be sacrifice in order to meet the timeliness requirements of a real-time (or near real-time) application. Second, we provided results from our simulation experiments, in which the simplifying assumptions necessary for our analytical results were lifted. Our simulation results confirm the ability of TCP Boston in reducing response time through a minute increase in bandwidth requirements. This is valuable for real-time applications, especially when the increase in bandwidth requirements can be tolerated.

Future Work:
Our implementation of TCP Boston (and simulations thereof) does not fully exploit the protocol's dynamic redundancy control features. In particular, in all our experiments, the level of spatial redundancy (used to mask the effect of individual cell losses at a congested switch) was kept static. By adjusting the amount of redundancy employed in TCP Boston to various parameters (*e.g.* deadline slack factor, level of congestion, advertised worst-case cell loss rates for a particular class of service, *etc.* we anticipate that TCP Boston's performance would improve even more than what we have presented in this paper. We are currently studying the premise of implementing such improvements.

Our work in progress involves the implementation of a version of TCP Boston that does not require our current (minor) modification of the AAL5 layer. Also, we are trying to find a way of improving our implementation of the IDA dispersal and retrieval algorithms to reduce (or eliminate) the buffer space requirement at the sender and receiver ends. This is particularly important in high bandwidth-delay product network environments. Finally, we plan on deploying TCP Boston in an experimental ATM switching environment, which is currently being installed in our department.

References

[1] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing Reference Locality in the WWW. In *Proceedings of IEEE PDIS'96: The International Conference in Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[2] ANSI. AAL5 – A New High Speed Data Transfer AAL. In *ANSI T1S1.5 91-449*. November 1991.

[3] G. Armitage and K. Adams. Packet Reassembly During Cell Loss. *IEEE Network Mag.*, 7(5):26–34, September 1993.

[4] R. Atkinson. Default IP MTU for use over ATM AAL5. In *RFC 1626*. May 1994.

[5] Azer Bestavros. IDA-based disk arrays. Technical Memorandum 45312-890707-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, July 1989.

[6] Azer Bestavros. SETH: A VLSI chip for the real-time information dispersal and retrieval for security and fault-tolerance. In *Proceedings of ICPP'90, The 1990 International Conference on Parallel Processing*, Chicago, Illinois, August 1990.

[7] Azer Bestavros. An adaptive information dispersal algorithm for time-critical reliable communication. In Ivan Frisch, Manu Malek, and Shivendra Panwar, editors, *Network Management and Control, Volume II*. Plenum Publishing Corporation, New York, New York, 1994.

[8] Azer Bestavros. AIDA-based real-time fault-tolerant broadcast disks. In *Proceedings of RTAS'96: The 1996 IEEE Real-Time Technology and Applications Symposium*, Boston, Massachusetts, May 1996.

[9] Azer Bestavros and Gitae Kim. TCP Boston. Technical Report BUCS-TR-96-014, Boston University, Computer Science Department, July 1996.

[10] Azer Bestavros and Gitae Kim. TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of Infocom'97: The IEEE International Conference on Computer Communication*, Kobe, Japan, April 1997.

[11] A. Bianco. Performance of the TCP Protocol over ATM Networks. In *Proceedingds of the 3rd International Conference on Computer Communications and Networks*, pages 170–177, San Francisco, CA, September 1994.

[12] Ernst Biersack. Performance Evaluation of Forward Error Correction in ATM Networks. *Comm. of ACM*, pages 248–257, August 1992.

[13] Lawrence Brakmo, Sean O'Maley, and Larry Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. Technical Report TR 94 04, The University of Arizona Computer Science Department, Tucson, AZ 85721, Febrary 1994.

[14] Charles Henry Brase and Corrinne Pellillo Brase. *Understandable Statistics*. D.C.Heath and Company, 1993.

[15] Thomas Chen and Stephen Liu. *ATM Switching System*. Artech House, Inc., 685 Canton St., Norwood, Ma 02062, 1995.

[16] Douglass E. Comer. *Internetworking with TCP/IP*, volume 1. Prentice Hall Inc., Englewood Cliffs, NJ, 1995.

[17] The Merit Corporation. Internet Resource Discovery Service Packets on NSFNET by packets. Available from `http://www.mids.com`, December 1994.

[18] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of WWW Client-based Traces. Technical Report BUCS-TR-95-010, Boston University, Computer Science Department, April 1995.

[19] Sally Floyd. Simulator Tests. Available in `ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z`. ns is available at `http://www-nrg.ee.lbl.gov/nrg`., July 1995.

[20] ATM Forum. *ATM User-Network Interface Specification*. Pretice Hall, Inc, Englewood Cliffs, New Jersey 07632, 1993.

[21] D. Le Gall. A video compression standard for multimedia applications. *Comm. of ACM*, 34(4):46–58, April 1991.

[22] M. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of ACM SIGCOMM'94*, London, UK, August 1994.

[23] Garth A. Gibson and David A. Patterson. Designing disk arrays for high data reliability. *Journal of Parallel and Distributed Computing*, 17(1-2):4–27, January/February 1992.

[24] M. Hassan. Impact of Cell Loss on the Efficiency of TCP/IP over ATM. In *Proceedings of the 3rd International Conference on Computer Communications and Networks*, pages 165–169, San Francisco, CA, September 1994.

[25] Yuh-Dauh Lyuu. Fast fault-tolerant parallel communication and on-line maintenance using information dispersal. Technical Report TR-19-1989, Harvard University, Cambridge, Massachusetts, October 1989.

[26] Sonu Mirchandani and Raman Khanna, editors. *FDDI Technology and Applications*. John Wiley & Sons, Inc., 1993.

[27] Kihong Park, Gitae Kim, and Mark E. Crovella. The Effects of Traffic Self-Similarity on TCP Performance. Technical report, Boston University Computer Science Department, 1996.

[28] J. Postel. Transmission Control Protocol. In *RFC 793*. September 1981.

[29] Michael O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[30] Krithi Ramamritham. Real-time databases. *International journal of Distributed and Parallel Databases*, 1(2), 1993.

[31] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communication*, 13(4):633–641, May 1995.